UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER OF PATENTS AND TRADEMARKS
Washington, D.C. 20231
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/602,713	06/26/2000	Richard A. Ross	68037526-2000	2885

7590 03/24/2003

Hewlett-Packard Company
Legal Department
Mail Stop 79
3404 E. Harmony Road
Fort Collins, CO 80529-9599

EXAMINER
ALAM, HOSAIN T

ART UNIT	PAPER NUMBER
2172	

DATE MAILED: 03/24/2003

Please find below and/or attached an Office communication concerning this application or proceeding.

UNITED STATES PATENT AND TRADEMARK OFFICE

COMMISSIONER FOR PATENTS
UNITED STATES PATENT AND TRADEMARK OFFICE
WASHINGTON, D.C. 20231
www.uspto.gov

BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

Paper No. 14

Application Number: 09/602,713
Filing Date: June 26, 2000
Appellant(s): ROSS, RICHARD A.

Gregory W. Osterloth, Reg. No. 36,232
For Appellant

EXAMINER'S ANSWER

This is in response to the appeal brief filed on January 27, 2003, in Paper No. 13.

(1) Real Party in Interest

A statement identifying the real party in interest is contained in the brief.

(2) Related Appeals and Interferences

A statement identifying the related appeals and interferences which will directly affect or be directly affected by or have a bearing on the decision in the pending appeal is contained in the brief.

(3) Status of Claims

The statement of the status of the claims contained in the brief is correct.

(4) Status of Amendments After Final

The after-final amendment filed on November 4, 2002, in Paper No. 10 was considered; the advisory action indicated that the amendment would be entered upon filing appeal.

Examiner's comments in the advisory action were as follows:

(1) In page 4, par. 1, the Applicant indicates that his invention is used for transmitting or transferring codes, but not for efficient execution of codes, However, rejected claims do not recite transmission/transfer of codes.

(2) In page 2-3, the applicant mischaracterizes the teachings of the Hamby reference by quoting the sections of choice, which appears to be a piecemeal analysis.

(3) The applicant recites the whole claim 22 and does not refer to any specific limitation to which the arguments are directed.

(5) Summary of Invention

The summary of invention contained in the brief is deficient because it paraphrases the claim language and does not simplify invention for a better understanding.

(6) Issues

The appellant's statement of the issues in the brief is correct.

(7) Grouping of Claims

The rejection of claims 22-23 stand or fall together because appellant's brief does not include a statement that this grouping of claims does not stand or fall together and reasons in support thereof. See 37 CFR 1.192(c)(7).

(8) Claims Appealed

The copy of the appealed claims contained in the Appendix to the brief is correct.

(10) Grounds of Rejection

The following ground(s) of rejection are applicable to the appealed claims:

Claims 22-23 are rejected under 35 U.S.C. 102(e). This rejection is set forth in prior Office Action, Paper No. 5.

Claim 22 and 23 are rejected under 35 U.S.C. 102(e) as being anticipated by U. S. Patent No. 5,848,274 issued to Hamby et al. ("Hamby")..

With respect to claim 22, Hamby teaches a method of resolving computer code having a plurality of types of code structures (col. 4, lines 47-50; "...bytecode representation of the program, which is both compact and target-independent, and than have an interpreter at the client execute the bytecode"), each type of code structures including a plurality of index references (col. 5, lines 39-40; "the code objects and

intermediate symbol or IL symbols are stored in a resident persistent table") as claimed comprising the steps of reading a list of identifiers for each type of code structure (col. 5, lines 54-56; "the Incremental Imager which forms the program image from code objects and their respective IL symbols stored in a persistent symbols table"), each list including an index reference corresponding to each of the identifiers in the list; and Replacing each of the index references in the computer code with the respective identifier corresponding to each respective index reference (col. 5, lines 36-39; "an incremental builder resident in the system is invoked which translates the source code files into code objects and their corresponding intermediate language (IL) symbols").

As to claim 23 (the method of claim 22, wherein the types of code structures comprise classes, methods, and fields; and reading the list of identifiers for each type of code structure comprises reading a list of classes, a list of methods, and a list of fields), Hamby teaches byte codes such as Java (col. 9, lines 16-17) and Java class files (col. 28, line 37) as code objects.

(11) Response to Argument

Appellant argues:

(1) Hamby excerpts set forth on page 4 of the brief accurately represent the teachings of Hamby. See page 4 of the brief.

(2) "If , as the Examiner asserts, claim 22's "index references" are to be equated with Hamby's IL symbols, then claim 22's "identifiers" would have to correspond to Hamby's code objects." See the bottom of page 5 and the top part of page 6, Brief.

(3) As to the teachings of Hamby, "At most, Hamby discloses replacing an IL symbol with the actual code the symbol represent. Hamby does not disclose replacing any sort of identifier." See page 6, paragraph 2.

(4) Hamby does not teach "a list of identifiers for each type of code structure" because the symbol table in Hamby is a specialized type of symbol table containing code objects that is fully translated machine language implementation of a function definition and intermediate language symbols. See page 7, 2nd paragraph.

(5) Hamby does not teach " a list of identifiers for each type of code structure." Page 7, 2nd paragraph.

Prior to responding to appellant's arguments, the examiner presents the position he has taken with respect to the finally rejected claims:

The invention as claimed:

22. A method of resolving condensed computer code having a plurality of types of code structures, each of the types of code structures including a plurality of index references, the method comprising the steps of:

reading a list of identifiers for each type of code structure, each list including an index reference corresponding to each of the identifiers in the list; and

replacing each of the index references in the computer codes with the respective identifier corresponding to each respective index reference.

The invention as interpreted by the examiner:

In accordance with MPEP 2111 (Claim Interpretation; Broadest Reasonable interpretation), during patent examination, the pending claims must be "given the broadest reasonable interpretation consistent with the specification." In re Prater, 415 F.2d 1393, 1404-05, 162 USPQ 541, 550-51 (CCPA 1969). See also In re Morris, 127 F.3d 1048, 1054-55, 44 USPQ2d 1023, 1027-28 (Fed. Cir. 1997) and In re Cortright, 165 F.3d 1353, 1359, 49 USPQ2d 1464, 1468 (Fed. Cir. 1999).

Given broadest reasonable interpretation, the scope (and/or "metes and bounds) of claim 22 is essentially a method of resolving condensed computer codes having a plurality of code structures. The invention provides an identifier to each of the code structures and uses an index to list the code structures and their respective identifiers. The code structures are replaced in the condensed computer codes by their respective identifiers.

For a better understanding of the condensed codes refer to applicant's disclosure, page 4, lines 11-20:

"The index listings contain listings of identifiers corresponding to the particular instances of the respective code structures occurring within the bytecode and index references corresponding to each of the identifiers included in the listing. The bytecode is reduced in size by replacing the various identifiers appearing in the bytecode with the corresponding index references. In this way, for example, code structures are replaced with index references within the bytecode and an index containing the data structure is maintained. "

With reference to the above section of the Disclosure, claim 22 requires that the identifiers of code structures listed in an index replace instances of code structures in computer code.

Summary of the teachings of Hamby, the applied prior art:

Hamby builds one or more program images without utilizing traditional linkers. A program image is a compilation of program codes and comprises code objects. Hamby also teaches the use of an incremental builder that builds a program image incrementally and by utilizing code objects and a persistent IL symbol table. The persistent symbol table provides indexing between the IL symbols and code objects. The advantage provided by Hamby's incremental builder is that it compares the code objects of a new program with the existing code objects by utilizing the IL symbol table

and includes only the new code objects for building the program image. ***See col. 25, lines 32-37***.

Other relevant teachings of Hamby:

Col. 5, lines 36-45
an incremental builder resident in the system is invoked which translates the source code files into code objects and their corresponding intermediate language (IL) symbols. The code objects and IL symbols are stored in a resident persistent symbol table. A resident incremental imager utilizes the code objects and IL symbols to form run-time code objects. The incremental imager then links together run-time code objects forming a program image which is executable.

Col. 6, lines 8-11
Code objects and IL symbols are low level data structures used to create an image, i.e., the "program"

Col. 6, lines 22-27
Incremental Builder of the present invention, which first determines if the source code has been previously compiled and saved in a persistent symbol table

Col. 11, lines 55-56
Symbol Table: The symbol table is the central and fundamental repository of information

Col. 12, lines 48-57
Code Objects: A code object represents a function or data definition as a sequence of bytes in the form required by the CPU architecture of the target processor. A code object includes a set of relocations, each of which specifies that a pointer value within the code object must point to the code object that is the definition of a specific IL symbol in order to load the code object as part of an executable program. In the system of the present invention, code objects and intermediate language symbols are stored in persistent memory as a persistent symbol table.
Col. 14, lines 1-12
Run-time Code Objects: As previously discussed, code objects are stored persistently, and are used to form the program image. References between code objects are not in a form suitable for execution on computer hardware. These references must therefore be translated again. In order not to have to modify the code objects for the purpose of forming a runnable program image, and then have to restore to the code objects their original contents, this invention makes copies of them in non-persistent memory, where the program image is formed. These non-persistent code objects are termed run-time code objects.

Col. 24, line 66 through col. 25, lines 38
The Imager/Debugger not only produces information for the hardware, it also produces information for the Runner. In that process, it need not worry about relocation forms and other complexities, since all the Imager/Debugger needs is to make sure to use addresses from category c whenever it communicates with the Runner. In some circumstances where space is of concern, and it is necessary to avoid duplication of information, the Runner will in fact accept addresses from category b, and itself translate them to category.
FIGS. 13 through 17 illustrate a preferred embodiment of incremental image formation according to the principles of the present invention. Referring now to FIG. 13, the use of the code objects and IL symbols in forming the program image is shown. An IL symbol, previously stored in the persistent symbol table, is

shown at 2000. Each IL symbol 2000 contains a definition field 2001 which comprises a code object pointer 2004 which points to a corresponding code object 2010 by means of direct memory addressing. Definition field 2001 is defined by IL instruction stream 1999. IL symbol 2000 further comprises a dependents field 2002 and an old definition field 2003. Optionally, IL symbol 2000 further defines an IL name 2005. It is important to note that name 2005 is used only to display information to the programmer, e.g., in debug mode, and is defined during image creation. Each IL symbol 2000 is formed by the incremental builder of the present invention from part of the IL stream 1999.
Code object pointer 2004 points to a code object 2010 corresponding to IL symbol 2000. Each code object 2010 contains a size field 2011, a relocations field 2012 and a code section, 2013. Code sections 2013 contain the previously discussed fully translated machine-language implementations of function definitions and the initial values of variables. Relocations field 2012 defines a relocations pointer 2014, which points to a relocations array 2020. Relocations array 2020 comprises a vector of relocation offsets and IL symbol references.
The IL symbol to which relocations pointer 2014 points may be the same IL symbol which invoked the formation of code object 2010, as in the case of a recursive process, or in the more typical case, may point to a different IL symbol altogether.
The update table is maintained by the Builder for the Imager, so that following editing or other changes to the source file, the Imager can locate the code objects which have changed. This ensures that Imager forms the image according to the newest code objects when updating the image.
The update table is maintained by the Builder for the Imager, so that following editing or other changes to the source file, the Imager can locate the code objects which have changed. This ensures that Imager forms the image according to the newest code objects when updating the image.

Appellant's arguments summarized:

Appellant appears to be arguing that Hamby does not teach the step of "replacing each of the index references in the computer codes with the respective identifier..." as recited in claim 22 because Hamby's IL symbols correspond to "code objects" that are "fully translated machine language implementation of a function definition", but not the "code structures" as claimed.

Examiner's Response to each of the arguments:

Argument (1):

Hamby excerpts set forth on page 4 of the brief accurately represent the teachings of Hamby. See page 4 of the brief.

Response:

The applicant has referred to the following excerpt:

an incremental byte code compiler for generating IL symbols and code objects from a byte code source file, a persistent symbol table for storing the IL symbols and code objects

While the above section may be an accurate depiction of Hamby teachings, there are other teachings that must be taken into account. For instance, Col. 5, lines 36-45; Col. 6, lines 8-11; Col. 6, lines 22-27; Col. 11, lines 55-56; Col. 12, lines 48-57; Col. 14, lines 1-12, and most notably Col. 24, line 66 through col. 25, lines 38

"The update table is maintained by the Builder for the Imager, so that following editing or other changes to the source file, the Imager can locate the code objects which have changed. This ensures that Imager forms the image according to the newest code objects when updating the image."

Therefore the appellant has failed to consider the reference as a whole as a person of ordinary skill or one of skilled in the art would have done.

Argument (2):

"If , as the Examiner asserts, claim 22's "index references" are to be equated with Hamby's IL symbols, then claim 22's "identifiers" would have to correspond to Hamby's code objects." See the bottom of page 5 and the top part of page 6, Brief.

Response: The appellant has attempted to map the claimed limitations to relevant teachings of Hamby. However, the appellant could not map them to appropriate teachings.

Some important limitations/terms of claim 22 are highlighted for convenience:

22. A method of resolving **condensed computer code** having a plurality of types of **code structures**, each of the types of code structures including a plurality of index references, the method comprising the steps of:

reading a list of **identifiers for each type of code structure**, each list including an **index reference corresponding to each of the identifiers** in the list; and

replacing each of the index references in the computer codes with the **respective identifier** corresponding to each respective index reference.

Hamby teaches:
condensed computer code as "program image" in Col. 14, lines 1-12
code structures as "code objects" Col. 6, lines 8-11
identifiers for each type of code structure, in conjunction with the IL symbols and code objects Col. 12, lines 48-57
index reference corresponding to each of the identifiers because the "Incremental Builder" uses an "update table" to obtain as to which "code objects" are to be updated during incremental program image generation. Col. 25, lines 32-37.
replacing each of the index references in the computer codes with the **respective identifier** because Hamby's "Incremental Builder" does not update all code objects but selective ones. Col. 25, lines 32-37.

The examiner recognizes that Hamby does not explicitly indicate the word "index", but it inherently teaches the indexing. For instance, in the following section, Hamby's utilizes an update table to locate the newest code objects. This "update table" with the "IL symbol table" provides the indexing scheme encompassed by claim 22.

"The update table is maintained by the Builder for the Imager, so that following editing or other changes to the source file, the Imager can locate the code objects which have changed. This ensures that Imager forms the image according to the newest code objects when updating the image."
The fact that Hamby does not recite the term "index" does not prevent it from inherently anticipating the indexing scheme as claimed.

Reference: is made to MPEP 2144.01 - Implicit Disclosure. "[I]n considering the disclosure of a reference, it is proper to take into account not only specific teachings of the reference but also the inferences which one skilled in the art would reasonably be expected to draw therefrom." In re Preda, 401 F.2d 825, 826, 159 USPQ 342, 344 (CCPA 1968).

Furthermore, it has been held that "**[t]o establish inherency, the extrinsic evidence must make clear that the missing descriptive matter is necessarily present in the thing described in the reference, and that it would be so recognized by persons of ordinary skill.**"

Schering Corp. v. Geneva Pharmaceuticals Inc., 64 USPQ2d 1032 (DC NJ 2002) Decided August 8, 2002. The prior art disclosure need not be express in order to anticipate. Even if a prior art inventor does not recognize a function of his or her process, the process can anticipate if that function was inherent. To establish inherency, the extrinsic evidence must make clear that the missing descriptive matter is necessarily present in the thing described in the reference, and that it would be so recognized by persons of ordinary skill. Inherency is not necessarily coterminous with the knowledge of those of ordinary skill in the art. Artisans of ordinary skill may not recognize the inherent characteristics or functioning of the prior art. However, the discovery of a previously unappreciated property of a prior art composition, or of a scientific explanation for the prior art's functioning, does not render the old composition patentably new to the discoverer. Insufficient prior understanding of the inherent properties of a known composition does not defeat a finding of anticipation.

Argument (3):

As to the teachings of Hamby, "At most, Hamby discloses replacing an IL symbol with the actual code the symbol represent. Hamby does not disclose replacing any sort of identifier."

Response:

The examiner recognizes that Hamby does not explicitly indicate the word "identifier", but it inherently teaches such identifier in the persistent IL symbol table because a table uses some kind of identifiers to refer to its stored element. Note that the incremental imager of Hamby utilizes the IL symbol table to form a program image. Without an identifier an element in the IL symbol table cannot be referenced.

See Col. 5, lines 36-45: "...an incremental builder resident in the system is invoked which translates the source code files into code objects and their corresponding intermediate language (IL) symbols. The code objects and IL symbols are stored in a resident persistent symbol table. A resident incremental imager utilizes the code objects and IL symbols to form run-time code objects. The incremental imager then links together run-time code objects forming a program image which is executable."

Argument (4):

Hamby does not teach "a list of identifiers for each type of code structure" because the symbol table in Hamby is a specialized type of symbol table containing code objects that is fully translated machine language implementation of a function definition and intermediate language symbols.

Response: While Appellant states that the code objects of Hamby are not the code structures as claimed, he does not point to the claim limitations he is relying on. Note that claim 22 recites "condensed computer codes" and "code structure." The appellant has provided no evidence that the "code structures" are not "fully translated machine language implementation of a function definition and intermediate language symbols."

Reference is made to the parent file of current application, 09/053,260, issued as U. S. Patent No. 6,163,780. The '870 patent recites "executable computer code". Current applicant does not indicate as to whether a "code structure" is executable or not.

In response to applicant's argument the examiner notes that a recitation of the intended use of the claimed invention must result in a structural difference between the claimed invention and the prior art in order to patentably distinguish the claimed invention from the prior art. If the prior art structure is capable of performing the intended use, then it meets the claim. In a claim drawn to a process of making, the intended use must result in a manipulative difference as compared to the prior art. See *In re Casey*, 152 USPQ 235 (CCPA 1967) and *In re Otto*, 136 USPQ 458, 459 (CCPA

1963). Claim 22 requires that a code structure be replaced regardless of its kind. In other words, the type of a code structure does not control how it is replaced.

Argument (5):

Hamby does not teach " a list of identifiers for each type of code structure."

Response: Hamby teaches Java "class files" as code objects. See col. 28, line 37 through col. 29, line 36. The list of identifiers are essentially the instances of class files.

As to appellant's failure to understand "how Hamby's teachings that an incremental imager supports Java has any bearing on the allowability of claim 23" (Brief, page 7, lines 5-6), the examiner request the appellant to review the following: Hamby builds one or more program images without utilizing traditional linkers. A program image is a compilation of program codes and comprises code objects. Hamby also teaches the use of an incremental builder that builds a program image incrementally and by utilizing code objects and a persistent IL symbol table. The persistent symbol table provides indexing between the IL symbols and code objects. The advantage provided by Hamby's incremental builder is that it compares the code objects of a new program with the existing code objects by utilizing the IL symbol table and includes only the new code objects for building the program image. ***See col. 25, lines 32-37***.

For the above reasons, it is believed that the rejections should be sustained.

Respectfully submitted,

Hosain T Alam
Primary Examiner
Art Unit 2172

March 21, 2003

Conferees:

Frantz Coby, Primary Examiner, AU 2171 FC

Jean M. Corrielus, Primary Examiner, AU 2172 Acte for J. C.

BAKER & MCKENZIE
805 THIRD AVENUE
NEW YORK, NY 10022